Opinnäytetyö (AMK)

Tietotekniikka

Sulautetut ohjelmistot

2014

Jami Koivisto

DIGITAALISEN SIGNAALINKÄSITTELIJÄN TOTEUTUS ARDUINOLLA

TURUN AMMATTIKORKEAKOULU
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tietotekniikan koulutusohjelma | Sulautetut ohjelmistot

2014 | 29

Tiina Ferm

Jami Koivisto

DIGITAALISEN SIGNAALINKÄSITTELIJÄN TOTEUTUS ARDUINOLLA

Opinnäytetyön tarkoituksena oli suunnitella ja toteuttaa Arduino-kehitysaslustaan perustuva harrastelijakäyttöön sopiva digitaalinen signaalinkäsittelijä. Työssä tehtiin järjestelmä, jossa Arduinolla muunnetaan analoginen signaali digitaaliseksi, jota sitten käsitellään tietokoneelle luodussa käyttöliittymässä. Arduinon ominaisuuksia testattiin työn kuluessa. Toteutetulle signaalinkäsittelijälle tehtiin myös kaksi yksinkertaista testitapausta, joissa samalla havainnoillistettiin signaalinkäsittelyn käyttöä.

Teoriaosuudessa perehdyttiin lyhyesti työn kannalta tärkeisiin digitaaliseen signaalinkäsittelyn menetelmiin ja teorioihin, kuten Fourier-muunnokseen ja digitaalisiin suodattimiin. Lisäksi käsiteltiin yleisesti Arduino-konseptia ja Arduino Uno -kehitysalustan ominaisuuksia ja toimintaa.

Työn tuloksena luotiin toimiva järjestelmä digitaaliseen signaalinkäsittelyyn. Järjestelmää on mahdollisuus tulevaisuudessa jatkaa lisäämällä siihen uusia ominaisuuksia. Valmis toteutus jäi omaan käyttöön erilaisten elektroniikkaprojektien tueksi.

ASIASANAT:

Arduino, mikrokontrolleri, AD-muunnin, signaalinkäsittely

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information Technology | Embedded Software

2014 | 29

Tiina Ferm

Jami Koivisto

IMPLEMENTATION OF A DIGITAL SIGNAL PROCESSING DEVICE WITH ARDUINO

The purpose of this thesis was to design and implement an Arduino-based digital signal processing system for a hobbyist. The system had Arduino which converts analog signal to digital and sends it to the computer to be processed. The features of Arduino were tested along the work and the final system had two test cases which demonstrated the usage of signal processing features.

The theoretical part of the thesis introduces digital signal processing methods and theories in the perspective of the work. It also gives an overview of Arduino Uno and the features and functions of the Arduino development platforms.

As a result, a working system for digital signal processing was created. The system has opportunities to be further developed in the future by adding new features. The finished implementation exists for personal use in various electronic projects.

KEYWORDS:

Arduino, microcontroller, AD-converter, signal processing

SISÄLTÖ

LIITTEET

KUVAT

TAULUKOT

KÄYTETYT LYHENTEET

AD-muunnos	Analogia-digitaalimuunnos.
AVR	Atmelin mikrokontrolleriperhe.
DFT	Diskreettiaikaisen jaksollisen signaalin Fourier-muunnos (Discrete Fourier Transform).
DTFT	Diskreettiaikaisen ei-jaksollisen signaalin Fourier-muunnos (Discrete-Time Fourier transform).
EEPROM	Uudelleenkirjoitettava muisti jossa data säilyy myös ilman virtaa (Electronically Erasable Programmable Read-Only Memory).
FFT	Nopea algoritmi diskreetin Fourier-muunnoksen laskemiseen (Fast Fourier Transform).
FIR	Ei-rekursiivinen digitaalinen suodatin (Finite Impulse Response).
Fourier	Ranskalainen matemaatikko ja fyysikko Jean Baptiste Joseph Fourier, jokakehiti mm. Fourier-analyysin.
IDE	Integroitu ohjelmointiympäristö (Integrated Development Enviroment).
IIR	Rekursiivinen digitaalinen suodatin (Infinite Impulse Response).
Java	Sun Microsystemsin luoma olio-ohjelmointikieli.
LGPL	Lisenssi, joka sallii ohjelman liittämisen eri lisenssin alaiseen ohjelmaan (Lesser General Public License)
PWM	Pulssinleveysmodulaatio, jossa jännitettä säädetään pulssisuhteen avulla (Pulse-Width Modulation)
USB	Sarjaväyläarkkitehtuuri, jonka kautta laitteet voivat kommunikoida keskenään (Universal Serial Bus).

1 JOHDANTO

Digitaalinen signaalinkäsittely on hyödyllinen työkalu erilaisissa elektroniikan projekteissa ja töissä. Tietokoneiden ja mikrokontrollereiden laskentatehon jatkuvasti kasvaessa sitä pystyy hyödyntämään yhä useampi niin elektroniikan parissa työskentelevä ammattilainen kuin vasta-alkajakin.

Työn tarkoituksena on toteuttaa harrastelijakäyttöön sopiva digitaalinen signaalinkäsittelijä Arduino-kehitysalustalla ja tietokoneella. Arduinolla muutetaan analoginen signaali digitaaliseen muotoon ja lähetetään tietokoneelle. Tietokoneelle luodaan käyttöliittymä, jossa vastaanotettua dataa voidaan visualisoida ja käsitellä digitaalisen signaalinkäsittelyn menetelmin, esimerkiksi suodattamalla ja laskemalla Fourier-muunnos.

Työssä käsitellään lyhyesti työn kannalta tärkeitä digitaalisen signaalinkäsittelyn menetelmiä ja teorioita. Lisäksi perehdytään yleisesti koko Arduino-konseptiin ja työhön valitun Arduino Uno -kehitysalustan ominaisuuksiin ja toimintaan.

Työ toteutetaan henkilökohtaiseen käyttöön erilaisten elektroniikan projektien tueksi. Tavoitteena on kehittää toimiva järjestelmä, jota pystyy tulevaisuudessa kehittämään eteenpäin uusia ominaisuuksia lisäämällä.

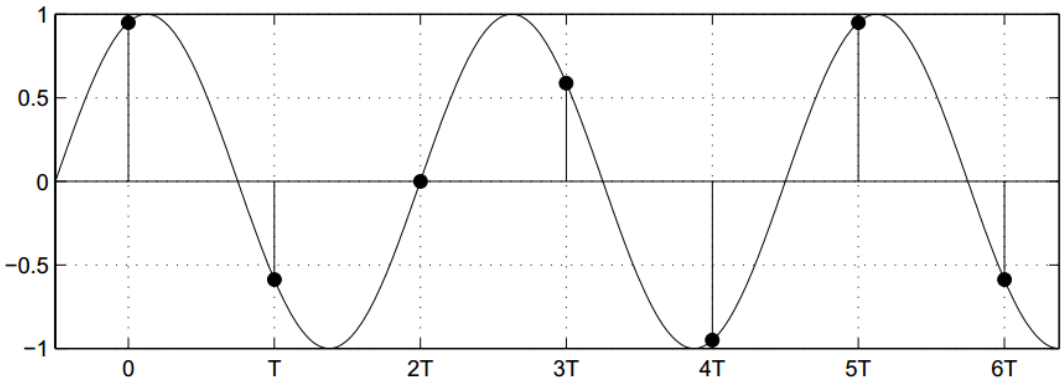
2 DIGITAALINEN SIGNAALINKÄSITTELY

Digitaalisen signaalinkäsittelyn tarkoituksena on esittää jokin signaali digitaalisessa muodossa ja analysoida, käsitellä tai irrottaa siitä tarpeellista informaatiota. Useimmiten signaalinkäsittelyssä signaalit ovat analogisia, joita näytteistetään tasavälein ja muutetaan digitaaliseen muotoon. Signaalinkäsittelyllä voidaan esimerkiksi poistaa signaalista tarpeetonta häiriötä, esittää se tarpeellisemmassa muodossa tai analysoida sitä erilaisin matemaattisin työkaluin. (Ifeachor 2002, 2.)

Tässä luvussa käsitellään lyhyesti erilaisia digitaalisen signaalinkäsittelyn menetelmiä keskittyen niihin, jotka ovat toteutettavan signaalinkäsittelijän kannalta keskeisiä.

2.1 Näytteistys ja kvantisointi

Analogista eli jatkuva-aikaista signaalia näytteistettäessä siitä otetaan näytteitä tasaisin ajanhetkin ja arvot talletetaan muistiin (Huttunen 2014, 3). Kuvassa 1 on esimerkki sinisignaalista, josta on otettu näytteitä ajanhetkillä *0*, *T*, *2T*, jne.



Kuva 1. Jatkuva-aikaisen signaalin näytteistys. (Huttunen 2014, 3.)

Vakio *T* kuvaa kahden näytteen välistä aikaa sekunteina. Sen käänteisarvo on näytteenottotaajuus *F*. Jos näytteenottotaajuus ei ole riittävä signaalille, tapahtuu laskostumista eli alinäytteistymistä. Näytteenottoteoreeman (Harry Nyquist 1928, Claude Shannon 1949) mukaan, jotta signaali voidaan rekonstruoida uudelleen, täytyy näytteenottotaajuuden olla vähintään kaksinkertainen suhteessa alkuperäisen signaalin suurimpaan taajuuteen. Tätä suurinta taajuutta kutsutaan Nyquistin taajuudeksi. Kaikki sitä suuremmat taajuudet laskostuvat, ellei niitä saada leikattua pois jollain järjestelmällä. (Huttunen 2014, 3–4.)

Signaalinkäsittelyssä näytteiden arvot täytyy esittää äärellisillä likiarvoilla eikä niillä voi olla ääretöntä tarkkuutta. Kvantisoinniksi kutsutaan muunnosta, jossa ääretön tarkkuus muunnetaan äärelliseen tarkkuuteen. Muunnoksen jälkeen syntyneiden tasojen välisiä eroja kutsutaan kvantisointiaskeleiksi. Kun kvantisointiaskeleet ovat tasaväliset, kvantisointi on lineaarinen. Esimerkiksi 10-bittisessä järjestelmässä on 2^{10} tasavälistä tasoa. (Jokinen 2003, 197.)

2.2 Digitaaliset suodattimet

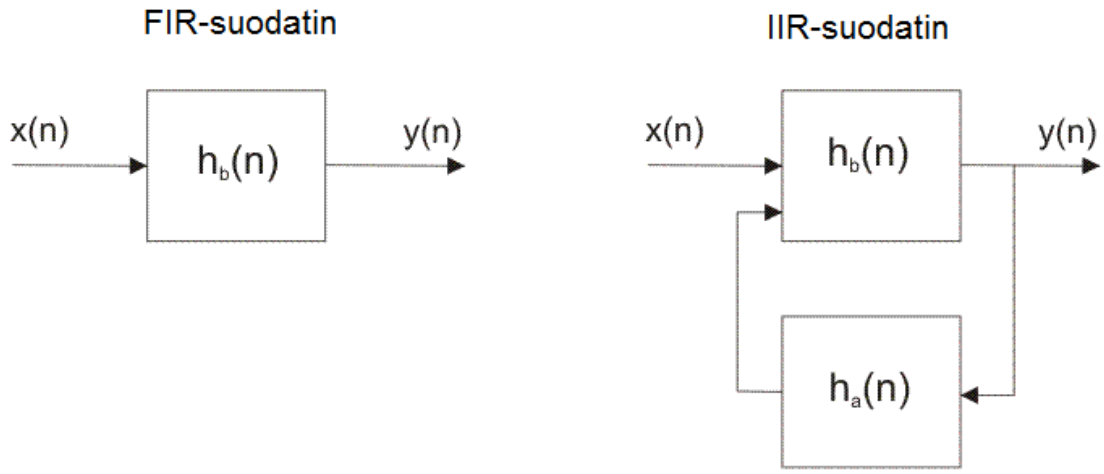
Digitaalinen suodattimen tarkoituksena on muuttaa signaalia siten, että se on sovellutusten kannalta hyödyllisemmässä muodossa. Se on erilaisten fyysisten komponenttien sijaan matemaattinen algoritmi, jolla voidaan esimerkiksi suodattaa signaalissa olevaa kohinaa pois säilyttämällä haluttu signaali mahdollisimman hyvin. Digitaalista suodatusta käytetään yleensä analogisiin signaaleihin. (Jokinen 2003, 274.)

Digitaalisilla suodattimilla on useita etuja analogisiin suodattimiin verrattuna. Esimerkiksi suodattimen ominaisuudet ja toiminta ovat täysin riippumattomia ympäristöolosuhteista, eivätkä niihin silloin vaikuta esimerkiksi lämmönvaihtelut. Suodattimet ovat myös hyvin tarkkoja, ja niiden suodattama data voidaan kerätä talteen helposti alkuperäisen datan kanssa. Niitä voidaan myös käyttää useammassa tulosignaalissa samanaikaisesti, ja niiden ominaisuudet on helppo monistaa. Suurin haitta digitaalisella suodattimella on sen nopeus, joka on

riippuvainen prosessorin nopeudesta ja ajasta muuttaa analoginen signaali digitaaliseksi ja suorittaa digitaalisen suodattamisen vaatimat aritmeettiset laskutoimitukset. (Jokinen 2003, 275–276.)

2.3 Digitaalisten suodattimien tyypit

Digitaaliset suodattimet voidaan jakaa yleisesti kahteen luokkaan: FIR-suodattimiin (finite impulse response) eli äärellisen impulssivasteen suodattimet ja IIR-suodattimiin (infinite impulse response) eli äärettömän impulssivasteen suodattimet. Impulssivasteella tarkoitetaan järjestelmän ulostuloa, kun sinne syötetään sisääntuloksi jokin impulssi. Kuvassa 2 on esitetty yksinkertaistettu lohkokaavio molemmista suodattimista. (Jokinen 2003, 276.)

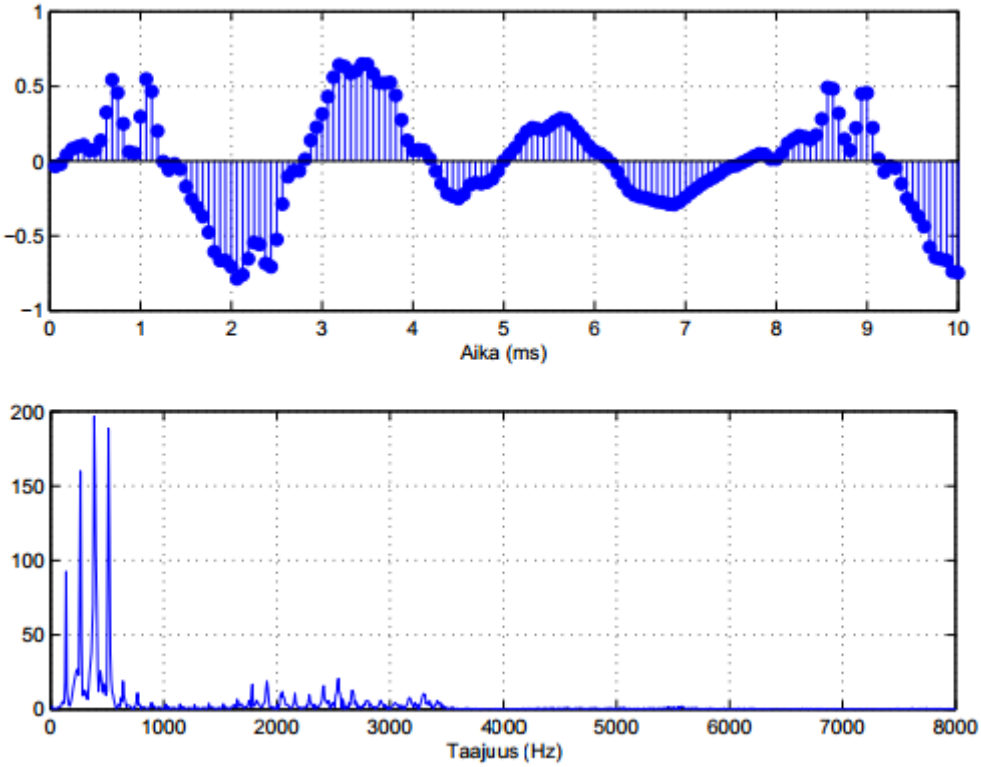


Kuva 2. FIR- ja IIR-suodattimen lohkokaaviot.

FIR-suodattimen ulostulo *y(n)* riippuu vain sisäänmenoista ja impulssivasteesta *h(n)*. Rekursiivisessa IIR-suodattimessa sen sijaan on takaisinkytkentä, jolloin sen ulostulo riippuu sisäänmenojen lisäksi myös aiemmin lasketuista ulostuloista. (Steven 1997a.)

2.4 Fourier-muunnos

Signaali voidaan esittää kahdella eri tasolla: aikatasolla ja taajuustasolla. Fourier-muunnos on näiden kahden tason välinen matemaattinen suhde. Fourier-muunnoksella voidaan kuvata signaalin sisältämiä taajuuksia, ja sen tuloksena saadaan taajuusjakauma. Kuvassa 3 on esimerkki Fourier-muunnoksesta. (Steven 1997b.)



Kuva 3. Signaali ja sille tehty Fourier-muunnos. (Huttunen 2014, 33.)

Signaalin jaksollisuudesta ja tyypistä riippuen Fourier-muunnoksesta käytetään eri nimityksiä. Jos muunnetaan ei-jaksollista jatkuva-aikaista (analogista) signaalia, käytetään nimitystä Fourier-muunnos. Signaalin ollessa jaksollinen ja jatkuva-aikainen nimitys on Fourier-sarja. Jos signaalina on ei-jaksollinen ja diskreettiaikainen (eli analogisesta digitaaliseksi muunnettu signaali, joka saa arvoja vain tietyillä ajanhetkillä), käytetään nimitystä diskreettiaikainen Fourier-

muunnos (DTFT). Kun signaali on jaksollinen ja diskreettiaikainen, muunnostyyppi on diskreetti Fourier-muunnos (DFT). (Huttunen 2014, 33.)

Työn kannalta kiinnostavin muunnos on diskreetti Fourier-muunnos. Koska analogista signaalia näytteistetään tasavälein ja tarkoituksena on saada lopputulokseksi äärellinen määrä taajuuksia, on DFT-muunnos kaikkein tarkoituksenmukaisin.

2.5 Nopea Fourier-muunnos

Nopea Fourier-muunnos eli FFT (Fast Fourier Transformation) on algoritmi, joka vähentää diskreettiin Fourier-muunnokseen kuluvaa aikaa. Jos signaalista otetaan *N* kappaletta käsiteltäviä pisteitä, nopea Fourier-muunnos laskee muunnokseen tarvittavien laskutoimitusten määrän arvosta N^2 arvoon $N \log N$. Ero on huomattava etenkin, kun pisteiden *N* määrä on suuri. (Weisstein 2014.)

3 ARDUINO

Arduino on avoimen lähdekoodin kehitysalusta, jonka ovat kehittäneet Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino ja David Mellis Italiassa 2005. Heidän tarkoituksenaan oli luoda helppokäyttöinen ja joustava laitteiston ja ohjelmiston yhdistelmä, joka olisi jokaisen tavoitettavissa. (Barrett 2013, 1.)

3.1 Arduino-konsepti

Arduino perustuu yksinkertaiseen mikrokontrollerialustaan ja ohjelmointiympäristöön. Arduinoa voidaan käyttää kehittämään interaktiivisia laitteita, lukemaan erilaisia kytkimiä ja antureita tai esimerkiksi kontrolloimaan valoja, moottoreita tai muita fyysisiä laitteita. (Arduino 2014c.)

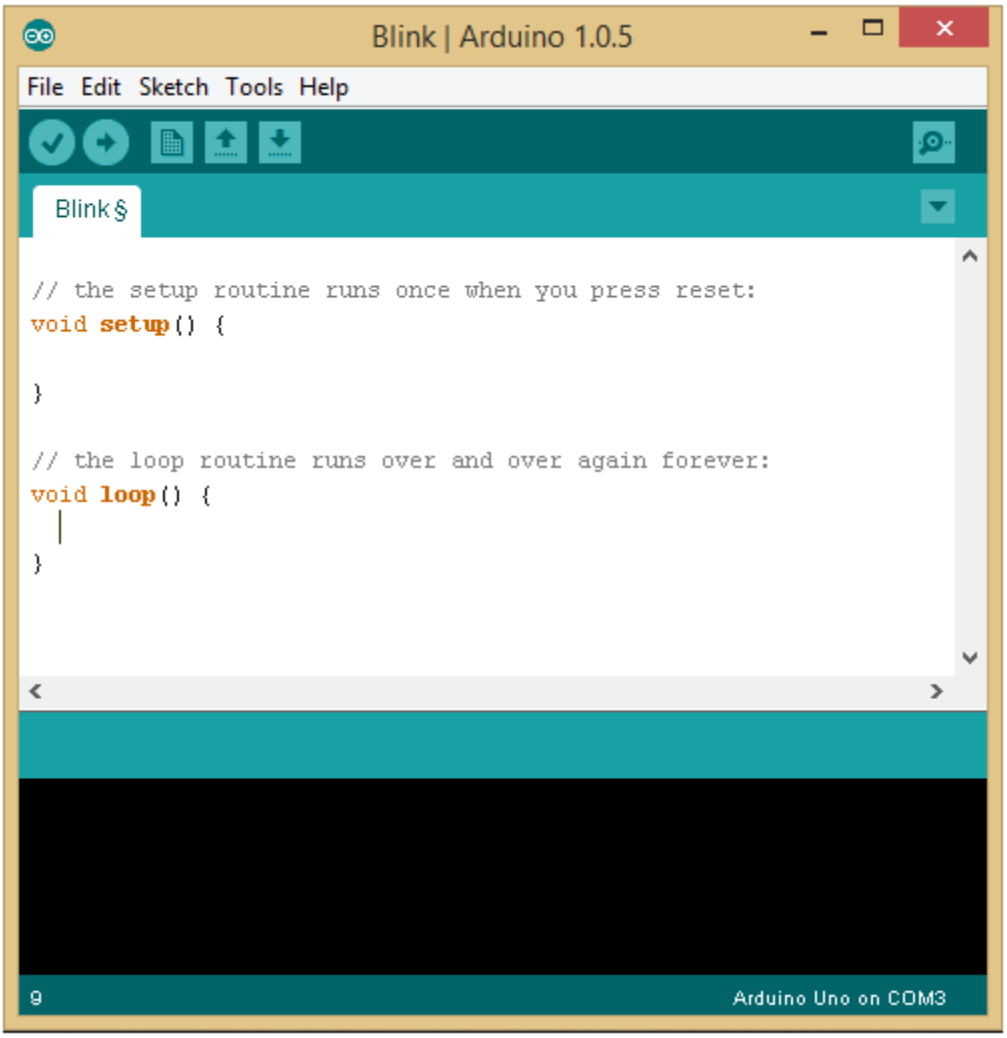
Arduino-konseptin tarkoituksena on yksinkertaistaa työskentelyä mikrokontrollereiden kanssa. Se tarjoaa myös muita huomattavia etuja esimerkiksi elektroniikan parissa työskentelevälle harrastelijalle:

- edullisuus
- käyttöjärjestelmästä riippumattomuus
- helppokäyttöinen ohjelmistoympäristö
- avoimen lähdekoodin ohjelmisto
- avoimen lähdekoodin laitteisto.

Arduino on halpa vaihtoehto vastaavanlaisissa tuotteissa. Laitteen hinta vaihtelee noin 20 € – 50 €. Arduinon ohjelmisto ei myöskään ole käyttöjärjestelmästä riippuvainen, vaan se toimii niin Windows-, Linux- kuin Macintosh OSX -käyttöjärjestelmissä. Arduino perustuu Atmelin ATmega8- ja ATmega168-mikrokontrollereihin, ja kaikkien alustojen piirilevyistä julkaistaan suunnitelmat ja piirustukset Creative Commons -lisenssillä. (Arduino 2014c.)

3.2 Ohjelmointiympäristö

Arduinon ohjelmointiympäristöä kutsutaan Arduino IDE:ksi (Kuva 4.). Sen yhtenä päätarkoituksena on helpottaa koodin kirjoittamista ja lataamista Arduino-laitteisiin. Arduino IDE on kehitetty Java-kielellä, ja se perustuu avoimen lähdekoodin ohjelmiin, kuten Processing ja avr-gcc. (Arduino 2014d.)



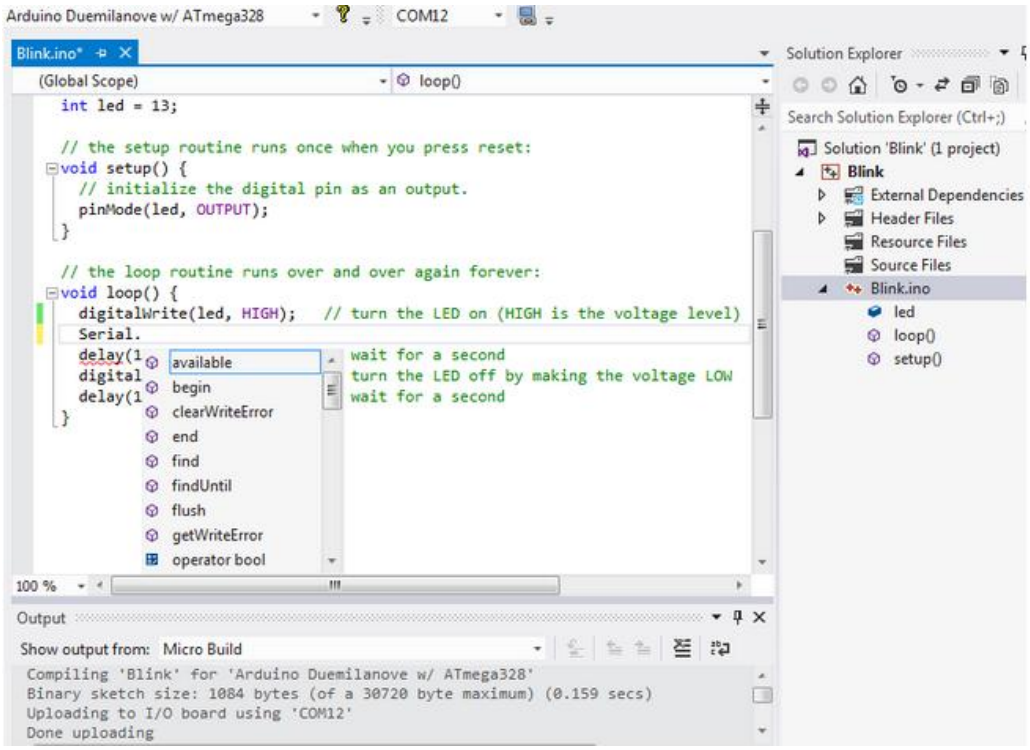
Kuva 4. Kuvankaappaus Arduino IDE:stä.

Arduinon ohjelmointikieltä voi laajentaa erilaisilla C++ -kirjastoilla. Koska Arduino-laitteet perustuvat Atmelin ATmega mikrokontrollereihin, voi ohjelmointikielenä käyttää myös AVR C -kieltä. (Arduino 2014c.)

3.3 Vaihtoehtoja Arduino IDE:lle

Arduino IDE:stä puuttuu sen yksinkertaisuuden vuoksi monia ominaisuuksia, joita kokeneempi käyttäjä voisi tarvita. Siinä ei ole esimerkiksi koodin simulointia, tarkempaa virhetilojen tarkastelua ja koodin täydennystä. Arduino tukee kuitenkin muitakin ohjelmointiympäristöjä, joista on seuraavaksi mainittu muutama.

Visual Micro on ilmainen ohjelmalisäke Microsoft Visual Studio 2008-2013 - ja Atmel Studio 6 -ohjelmille. Se tukee kaikkia samoja ominaisuuksia, kirjastoja ja kehitystyökaluja kuin Arduino-ohjelmointiympäristö. Visual Micro (Kuva 5.) tarjoaa näiden lisäksi käyttäjälle kehittyneemmän ja ammattimaisemman käyttöliittymän mm. automaattisen ohjelmavirheiden paikannuksen ja koodin täydennyksen avulla. (Visual Micro 2014.)



Kuva 5. Visual Micro ja Microsoft Visual Studio 2012. (Visual Micro 2014.)

The Arduino Eclipse Plugin on avoimen lähdekoodin ohjelmalisäke Eclipse-ohjelmointiympäristölle. Ohjelmalisäke tarjoaa useita ominaisuuksia, joita

Arduino IDE:stä puuttuu. Näitä ovat esimerkiksi koodin täydennys, tuki versionhallintaohjelmille, funktioiden dokumentointi ja virheiden havaitsemisen. (Eclipse.baeyens.it 2014.)

3.4 Arduino Uno

Arduino-tuoteperheen Uno R3 -alusta (Kuva 6) on kyseisen mallin kolmas versio ja tulevien muiden Arduino-alustojen referenssimalli. Uno R3:ssa on ATmega328-mikrokontrolleri, joka toimii 16 MHz:n kellotaajuudella. Siinä on 14 digitaalista I/O-pinniä, 6 analogista sisääntuloa ja 5 V:n ja 3,3 V:n pinnit. Uno R3:ssa on myös USB-liitäntä ja DC-virtaliitin, joita molempia voidaan käyttää virtalähteenä. Uno R3 toimii 5 V:n käyttöjännitteellä USB-liitännästä, mutta sitä voidaan myös käyttää erillisellä virtalähteellä, jolloin suositusrajat ovat 7–12 V. Alustassa on 34 kB Flash-muistia, 2 kB SRAM-muistia ja 1 kB EEPROM-muistia. (Arduino 2014a.)

Kuva 6. Arduino Uno R3 edestä kuvattuna. (Arduino 2014a.)

Arduino Uno R3:n jokaista 14 digitaalista pinniä voidaan käyttää sisääntulona ja lähtönä. Pinnit toimivat 5 V:n jännitteellä ja pystyvät syöttämään ja kestämään 40 mA:n tasavirtaa. Digitaalisista pinneistä 6 toimii myös 8-bittisinä PWM-

lähtöinä (Pulse-Width Modulation). Uno R3:n analogiset pinnit tarjoavat 10-bittisen resoluution eli palauttavat arvon 0:n ja 1 024:n väliltä. Pinnit operoivat oletuksena 0–5 V jännitteellä, mutta mitattavaa jännitettä voi muuttaa AREF-pinnillä. 5 V on samalla pinnien yläraja, jota ei saa ylittää. (Arduino 2014a.)

Arduino Uno R3:n digitaalisia pinnejä 0 ja 1 voidaan käyttää sarjaliikennettä varten. Toisin kuin edellisissä alustoissa, Uno R3:ssa on ATmega16U2-apuprosessori, joka toimii USB:n ja sarjaliitännän muuntimena. Käyttöä varten ei tarvitse asentaa ajureita vaan alusta näkyy tietokoneelle virtuaalisena sarjaporttina. (Arduino 2014a.)

4 TYÖN TOTEUTUS

Työn tarkoituksena oli toteuttaa toimiva signaalinkäsittelijä Arduinon ja PC-ohjelmiston yhdistelmällä. Työ toteutettiin kahdessa vaiheessa. Ensimmäiseksi suunniteltiin ja tehtiin Arduino-laitteelle ohjelmakoodi, jolla luotiin puitteet työn seuraavalle vaiheelle eli PC-ohjelmiston suunnittelulle ja toteutukselle.

Signaalinkäsittelijässä Arduinon varsinaisena tarkoituksena on muuttaa mitattava signaali digitaaliseksi ja lähettää se tietokoneelle USB-yhteyden kautta. Tietokoneella on tarkoitus tehdä suurempaa laskentatehoa vaativat algoritmit, kuten esimerkiksi diskreetti Fourier-muunnos ja digitaaliset suodattimet.

Tässä luvussa käydään läpi työn eri vaiheet. Työn toteutuksen kannalta keskeisin vaihe oli Arduino Uno R3:een tarkempi tutustuminen ja sen ohjelmointi ja testaus, minkä takia luvussa keskitytään niihin tarkemmin. PC-ohjelmiston toteutus käydään lyhyesti läpi painottaen työn kannalta tärkeitä aiheita. Lopullisen signaalinkäsittelijän testaamista esitellään luvussa 5.

4.1 Tarvikkeet ja ohjelmat Arduinolle

Työhön tarvittiin Arduino Uno R3 -kehitysalusta ja johtimia. Alustaa ei tarvinnut erikseen hankkia, kuten ei myöskään työssä käytetyttyjä johtimiakaan, vaan ne oli jo valmiiksi saatavilla.

Varsinainen ohjelmointi alustalle tehtiin Arduino IDE -ohjelmalla, koska Arduino-puolen ohjelmasta oli tarkoitus tulla mahdollisimman yksinkertainen. Tämän takia ohjelmoinnissa ei erikseen tarvittu esimerkiksi koodin täydennystä tai virhetilojen tarkastelua, joita yleensä suuremmissa projekteissa usein tarvitaan. Arduino IDE:ssä on myös erillinen Serial Monitor -ikkuna, jolla pystyy simuloimaan sarjamuotoisen datan lähettämistä ja vastaanottamista alustalta,

mikä helpottaa esimerkiksi ohjelmakoodin kokeilua ja erilaisten muuttujien tarkastelua.

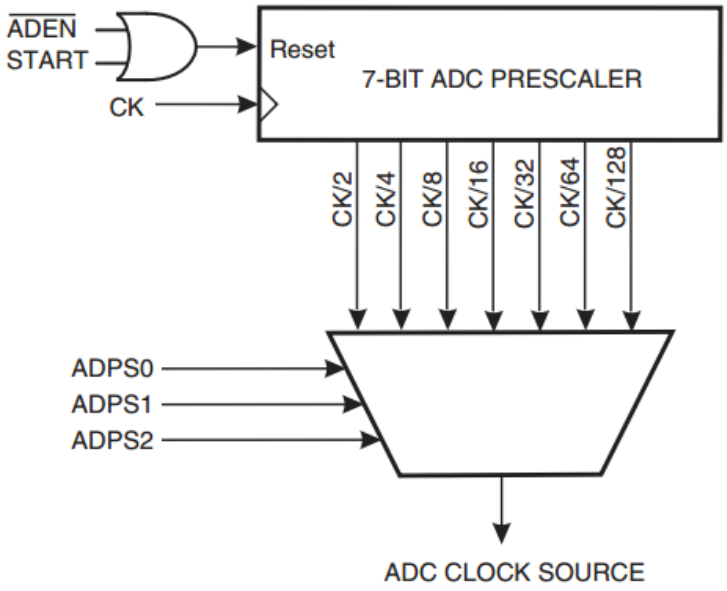
4.2 Arduinon ohjelmointi

Arduinon ohjelmointi aloitettiin suunnittelemalla ohjelmakoodin perusrakenne. Kun Arduinon laittaa päälle, se asettaa määrätyt arvot itselleen esimerkiksi sarjaliikenteelle ja AD-muunnokselle. Tämä jälkeen Arduino jää odottamaan PC-ohjelman käskyä aloittaa AD-muunnosten suorittaminen. Kun käsky on saatu, se lukee arvon pinnistä A0 (analoginen pinni 0). Lopuksi Arduino tulostaa saadun arvon sarjaporttiin. Arduino jää tämän jälkeen toistamaan lukemista ja tulostamista, kunnes se sammutetaan tai tietokoneelta annetaan käsky lopettaa. Jos tietokoneelta tulee käsky, Arduino jää odottamaan kunnes se saa käskyn jatkaa suoritusta uudelleen. Lopullinen ohjelmakoodi on liitteessä 1.

4.2.1 AD-muunnos

Varsinainen toteutus aloitettiin tutkimalla, kuinka suuri mittausnopeus Arduino Uno R3:lla voidaan saavuttaa. Tällä tavoin saatiin selville esimerkiksi minkälaisia signaaleja sillä voidaan tutkia ja kuinka tarkasti. Arduinon analogisten signaalien mittausnopeus on riippuvainen sen AD-muuntimen nopeudesta muuttaa analoginen signaali digitaaliseksi, minkä takia työn tämä vaihe oli Arduinon ohjelmoinnissa tärkein.

Arduino Uno R3:n ATmega328:n AD-muunnokseen kuluva aika riippuu suorittimen kellotaajuudesta ja ADPS-rekisteristä (ADC Prescaler Select), jolla määritetään esijakaja AD-muuntimen kellotaajuutta määrittäessä. AD-muuntimen (Kuva 7.) kellotaajuuden olisi sopivinta olla 50 kHz:n ja 200 kHz:n välillä hyvän tarkkuuden eli 10 bitin saavuttamiseksi. Oletuksena ATmega328:n esijakaja on arvossa 128, jolloin kellotaajuus on mikrokontrollerin kellotaajuuden ja esijakajan suhde eli 16 MHz / 128 = 125 kHz. (Atmel 2009, 253.)



Kuva 7. AD-muuntimen esijakaja. (Atmel 2009, 253.)

Arduinon ATmega328:n yksi AD-muunnos vie 13 kellopulssia. Muuntimen kellotaajuuden ollessa 125 kHz voidaan laskea sen maksimaalinen mittausnopeus: 125 kHz / 13 = ~9600 Hz. Työn tavoitteena oli kuitenkin saavuttaa tätä suurempi mittausnopeus, joten AD-muunnokselle laskettiin arvot myös muilla esijakajilla. AD-muunnin ei pysty operoimaan 1 MHz:ä suuremmilla kellotaajuuksilla, joten taulukkoon 1 on laskettu arvot esijakajilla 16, 32, 64 ja 128. (Atmel 2006, 10.)

Taulukko 1. Lasketut teoreettiset mittausnopeudet eri esijakajilla ja kellotaajuuksilla.

Esijakaja	AD-muuntimen kellotaajuus	Mittausnopeus
16	1 MHz	~76 900 Hz
32	500 kHz	~38 500 Hz
64	250 kHz	~19 200 Hz
128	125 kHz	~9600 Hz

AD-muuntimen suurin mittausnopeus on tällöin noin 77 000 näytettä/s. Kellotaajuus menee kuitenkin yli suositellun 200 kHz:n, jolloin muuntimen tarkkuutta täytyy pudottaa. Jos käytössä on 10 bitin tarkkuus (0–1023), AD-

muuntimella voidaan havaita minimissään 5V / 1023 = 0.0049 V:n eli 4,9 mV:n ero jännitteessä. Tarkkuuden ollessa 8 bittiä (0–255) jännite-ero on 5 V / 255 = 19,6 mV. Työssä päätettiin valita parempi mittausnopeus tarkkuuden sijaan. Esijakajaksi päätettiin valita 16 tai 32, jotka saadaan muokkaamalla ADCSRA-rekisterin (ADC Control and Status Register) ADPSx-bittejä. (Atmel 2009, 264.)

Tarkkuuden ollessa 10 bittiä AD-muuntimen tulos saadaan kahdessa osassa AD-muuntimen rekistereistä ADCL ja ADCH. Jos tarkkuutta kuitenkin vähennetään 8 bittiin, kuten työssä tehtiin, voidaan tulos lukea suoraan ADCH-rekisteristä. Tämä toteutetaan asettamalla ADLAR-bitti (ADC Left Adjust Result) arvoon 1 ADMUX-rekisterissä (ADC Multiplexer Selection Register). (Atmel 2009, 251.)

```
ADMUX |= (1 << ADLAR);
```

AD-muunnin voi muuntaa arvoja yksi kerrallaan, tai sitä voi käyttää vapaasti juoksevassa tilassa (Free running mode), jolloin AD-muunnin aloittaa uuden muunnon edellisen valmistuttua. Työn kannalta kätevämpi oli free running -tila, joka saadaan päälle asettamalla ADATE-bitti (ADC Auto Trigger Enable) arvoon 1 ADCSRA-rekisterissä. (Atmel 2009, 255.)

```
ADCSRA |= (1 << ADATE);
```

AD-muunnoksia voidaan hallita keskeytysten avulla. Kun ADIE-bitti (ADC Interrupt Enable) asetetaan arvoon 1, tapahtuu AD-muuntimen keskeytys aina kun edellinen on valmis. Muuntimen ollessa free running -tilassa keskeytys tapahtuu säännöllisesti. Kun tiedetään AD-muuntimen kellotaajuus ja muunnoksen kesto, voidaan muunnosten välinen aika laskea seuraavanlaisesti: 13 / 500 kHz = 0,0026 ms. (Atmel 2009, 251.)

Arduino Uno R3:n analogisia pinnejä voidaan käyttää digitaalisten pinnien tapaan. Tässä työssä sitä ominaisuutta ei kuitenkaan tarvittu, joten toiminnallisuus otettiin pois käytöstä käsittelemällä DIDR0-rekisteriä. Tämä vähentää samalla esimerkiksi virrankulutusta. (Atmel 2009, 249.)

4.2.2 Sarjaliikenne Arduinosta tietokoneeseen

Arduino-ohjelmoinnin seuraava vaihe oli yhteyden muodostaminen tietokoneeseen. Arduino Uno R3:n sarjaporttiasetusten määrittäminen on hyvin yksinkertaista. Serial.begin()-funktiolla laitteelle saadaan alustettua sarjaportti tietyllä sarjanopeudella. Esimerkiksi komento Serial.begin(9600) alustaa sarjaportin nopeudella 9 600 b/s. Oletuksena sarjaportti käyttää muotoa: 8 databittiä, ei pariteettia ja 1 lopetusbitti (stop bit). Asetuksia ei tämän työn puitteissa tarvinnut muuttaa. (Arduino 2014b.)

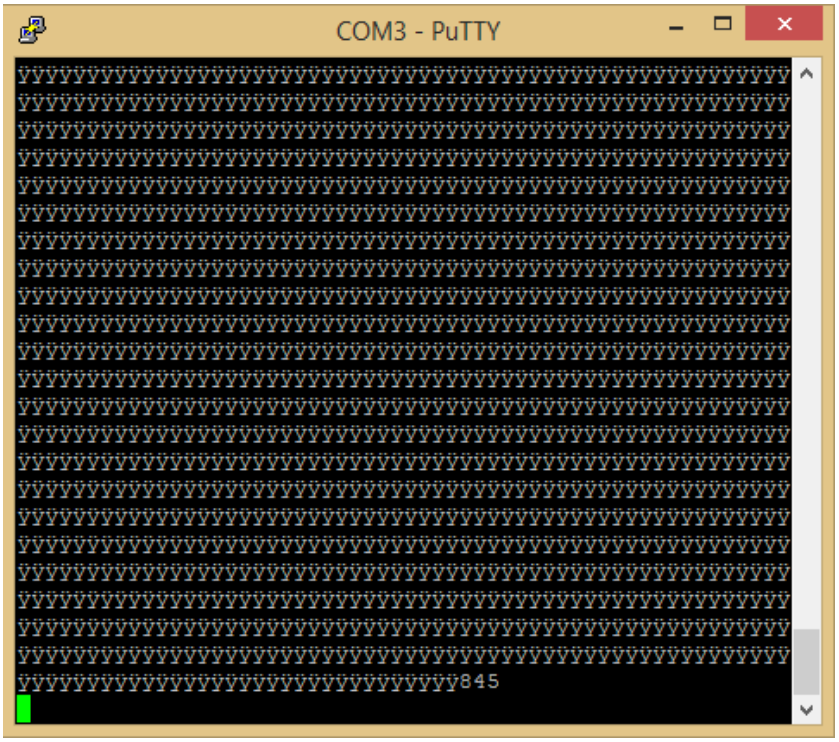
Arduino Uno R3:sta voidaan lähettää sarjamuotoista dataa tietokoneelle Serial.write()- ja Serial.print()-funktioilla. Työn kannalta näistä kahdesta write() oli käyttökelpoisempi, koska sillä dataa voidaan lähettää yksi tavu eli 8 bittiä kerrallaan. Tämä sopi hyvin AD-muuntimen 8 bitin resoluutioon, koska silloin yhtä mittaustulosta ei tarvitse lähettää useammassa osassa. Tämä helpotti työn PC-ohjelmiston määrittelyä ja ohjelmointia, koska tulevan datan vastaanottamisessa tarvitsi lukea tavu kerrallaan. Muussa tapauksessa lähetettävällä datalla olisi täytynyt olla tunnistettava muoto, esimerkiksi aloitusbitti, databitti, databitti, lopetusbitti. Tunnistettavalla muodolla olisi estetty väärien tulosten syntyminen, mutta mittausnopeus olisi kärsinyt huomattavasti.

4.2.3 Ohjelman testaaminen

Arduino-ohjelmakoodin testaamiseksi sille tehtiin lyhyt testiohjelma. Testiohjelman tarkoituksena oli testata erilaisia yhdistelmiä niin AD-muuntimen kellotaajuuden kuin sarjaportin nopeuksien välillä. Ohjelma oli hyvin yksinkertainen: toista 10 000 kertaa AD-muunto-lähetys ja tulosta tässä kulunut aika millisekunneissa. Liitteessä 2 on testiohjelman koko ohjelmakoodi.

Arduino IDE:n Serial Monitor -ikkuna tukee sarjanopeuksia vain 115 200 b/s saakka. Ohjelmaa haluttiin testata myös suuremmilla nopeuksilla, koska AD-muuntimen kellotaajuutta nostettiin. Tämän takia testaamisen apuna käytettiin PuTTY-ohjelmaa, joka on ladattavissa sivulta

http://www.chiark.greenend.org.uk/~sgtatham/putty. PuTTY-ohjelmaa voidaan käyttää Arduino IDE:n Serial Monitor -ikkunan tavalla valitsemalla Connection Type -kohdasta ”Serial” ja asettamalla sarjaportin ja -nopeuden oikein. Kuvassa 8 on esimerkki PuTTY-ohjelmaan tulostetusta testiohjelmasta, josta näkyy arvoja ascii-merkkeinä (ÿ = 255).



Kuva 8. PuTTY-ohjelmaan tulostettu testi.

Testiohjelman AD-muuntimen kellotaajuuden ja sarjaportin nopeuksia testattiin erilaisilla arvoilla ja tulokset kirjattiin ylös. Taulukossa 2 on esitelty testiohjelmasta saatuja tuloksia. Mittausnopeus laskettiin jakamalla kierrosten lukumäärä kuluneella ajalla, esimerkiksi 10000 / 0,845 = ~11 834. Sarjaportin nopeudet valittiin ATmega328:n datalehdestä löydetyillä arvoilla. (Atmel 2009, 203.)

Taulukko 2. Testiohjelmasta saadut tulokset.

Esijakaja	Sarjaportin nopeus	Kulunut aika (ms)	Mittausnopeus (näytettä/s)
32	115 200	845	11 834
32	230 400	447	22 371
32	345 600	297	33 670
32	500 000	286	34 965
32	1 000 000	284	35 211
16	115 200	843	11 862
16	230 400	447	22 371
16	345 600	341	29 325
16	500 000	340	29 412
16	1 000 000	340	29 412

Tulosten perusteella sarjaportin nopeudeksi valittiin 1 000 000 b/s ja esijakajaksi 32, koska esijakajalla 16 ei näyttänyt olevan merkitystä mittausnopeuteen. AD-muunnoksen teoreettinen nopeus esijakajalla 16 on noin 77 000 näytettä/s, mutta reaaliaikainen tulostaminen sarjaporttiin hidastaa muunnoksia selvästi. Hidastumista voisi vähentää käyttämällä puskurirakennetta, jolloin AD-muuntimen tuloksia kerättäisiin puskuriin ja sen ollessa täysi lähetettäisiin arvot sarjaporttiin. Työssä haluttiin kuitenkin mahdollisimman reaaliaikainen lähetys, joten muutoksia ohjelmakoodiin ei tehty.

Tulosten perusteella arvioitiin, että Arduino Uno R3 pystyy lähettämään noin 35 200 näyttettä/s. Nyquistin teorian mukaan kyseisellä näytteenottotaajuudella voidaan käsitellä noin 17,6 kHz:n taajuuksia. Tulosten perusteella Arduinon näytteenottotaajuus on riittävä moniin elektroniikan töihin. Se ei kuitenkaan ole tarpeeksi nopea korvaamaan esimerkiksi oskilloskooppia, jota käytetään yleisesti elektroniikan alalla.

4.3 PC-ohjelmisto

Työn toinen vaihe oli suunnitella ja toteuttaa PC-puolen ohjelmisto. Sen tarkoituksena on yksinkertaisen käyttöliittymän avulla kontrolloida Arduino Uno R3:sta ja vastaanottaa siltä tulevaa dataa yhtäjaksoisesti. Data oli tarkoitus piirtää käyttäjälle sopivaan muotoon mahdollisimman sujuvasti. Käyttöliittymässä piti myös olla mahdollisuus tarkastella Fourier-muunnoksen tuloksia erillisessä kuviossa ja suodattaa ruudussa olevaa dataa digitaalisella suodattimella.

4.3.1 Ohjelmiston valinta

Työssä käytettiin Microsoft Visual Studio 2012 -ohjelmankehitysympäristöä. Ohjelmisto valittiin, koska se oli ennestään tuttu eikä työssä ollut tarkoituksena tutustua muuhun vastaavaan ohjelmistoon. Ohjelmointikielenä Visual Studiossa käytettiin C#:a, ja ohjelmisto toteutettiin Microsoft Forms -sovelluksena.

Visual Studio tarjosi työhön tarvittavat työkalut ja ominaisuudet esimerkiksi sarjaliikenteelle, virheiden havaitsemiselle ja erillisille kirjastoille. Ohjelmistosta löytyy myös erilaisia piirto-ominaisuuksia oletuksena, mutta työhön päätettiin valita erillinen kirjasto datan visualisointia varten. Työssä käytettiin ZedGraph-kirjastoa, joka on C#:lla kirjoitettu erillinen luokkakirjasto 2-ulotteisten kuvioiden piirtämiseen. ZedGraph on LGPL-lisenssin (Lesser General Public License) alainen, joten se on täysin ilmaiseksi ladattavissa. ZedGraph-kirjastoa käytetään Visual Studio -projekteissa lisäämällä verkkosivuilta ladattava ZedGraph.dll-tiedosto projektin referenssitiedostoihin. Tämän jälkeen kirjaston funktioita voidaan käyttää Visual Studion omien funktioiden tapaisesti. (ZedGraphWiki 2007.)

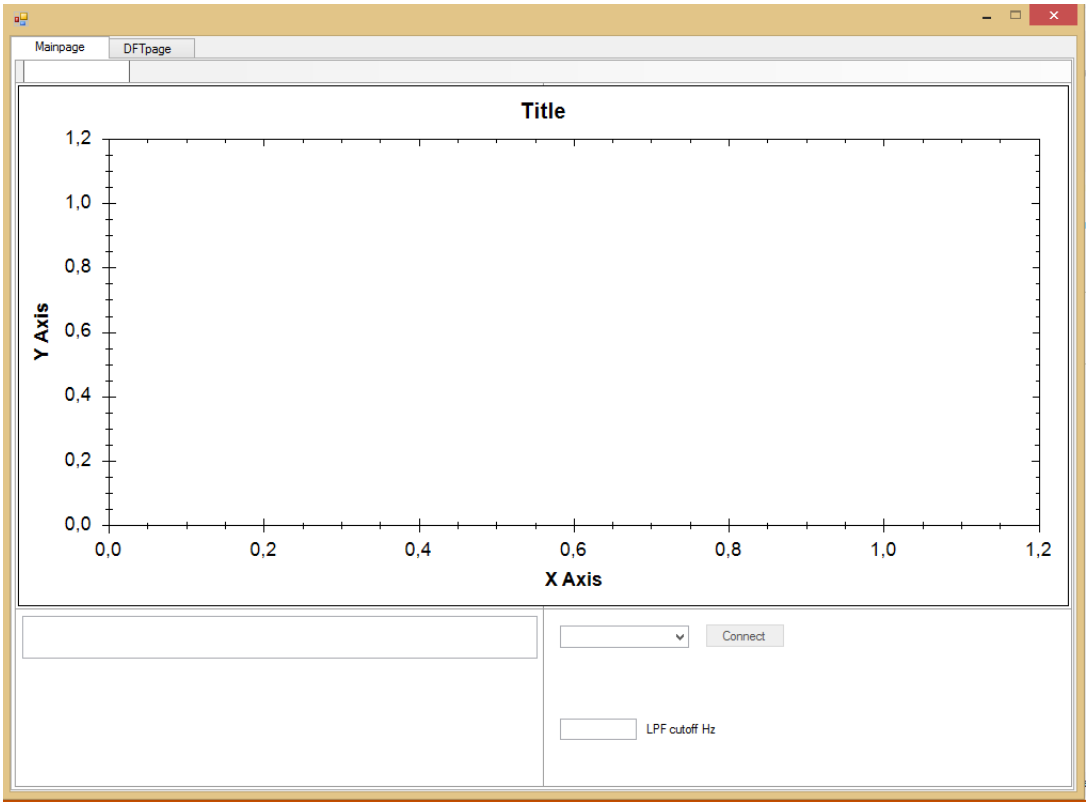
Työssä tarvittiin myös erillistä kirjastoa diskreetin Fourier-muunnoksen laskemiseksi. Tähän tarkoitukseen valittiin ALGLIB-kirjasto, joka on numeeriseen analyysiin ja datan prosessointiin tarkoitettu erillinen kirjasto. ALGLIB-kirjastoa voi käyttää Windows-, Linux- ja Solaris-käyttöjärjestelmillä ja

C++-, C#-, Pascal- ja Visual Basic -ohjelmointikielillä. Kirjasto otetaan käyttöön lisäämällä verkkosivuilta ladattava alglibnet2.dll-tiedosto referenssitiedostoihin. (ALGLIB 2014.)

4.3.2 Käyttöliittymän suunnittelu ja toteutus

Ohjelmiston toteuttaminen aloitettiin käyttöliittymästä ja siihen tulevista ominaisuuksista. Ohjelmiston oli tarkoitus tulla omaan käyttöön, minkä takia sen visuaalisiin ominaisuuksiin ei ollut syytä nähdä ylimääräistä vaivaa. Ohjelmistosta jätettiin tarkoituksella pois myös esimerkiksi erilaisia asetusvalikoita, koska niiden muuttaminen käy helposti myös ohjelmakoodin puolelta. Tärkeintä suunnittelussa oli käyttöliittymän helppokäyttöisyys ja yksinkertaisuus niin ulkonäöllisesti kuin ohjelmallisestikin.

Käyttöliittymä suunniteltiin kaksiosaiseksi: pääsivu, johon data piirretään, ja toinen sivu, johon lasketaan Fourier-muunnoksesta saadut arvot. Käyttöliittymän pääsivulle suunniteltiin piirrosalue, tekstikenttä virheentarkastelulle ja paneeli, johon sijoittaa erilaisia kontrolleja, kuten painikkeita ja muita tekstikenttiä. Toiselle sivulle tarkoituksena oli laittaa vain piirrosalue Fourier-muunnokselle. Suunnittelun jälkeen käyttöliittymä toteutettiin niiltä osin kuin oli suunniteltu (Kuva 9.).



Kuva 9. Toteutettu käyttöliittymä.

Käyttöliittymässä on kaksi sivua, jotka on toteutettu TabControl-luokalla. Keskellä oleva piirtoalue kuuluu ZedGraphControl-luokkaan, joka tarjoaa oletuksena käyttöön esimerkiksi lähentämisen, loitontamisen ja piirtoalueen tallentamisen kuvana. Lisäksi käyttöliittymässä on ylhäällä tekstikenttä näytteistystaajuuden havainnointiin ja alhaalla tekstikenttä virheilmoituksille. Sarjaportin valitsemiseksi ja yhteyden muodostamiseksi on myös lisätty omat kontrollinsa. Toisella sivulla on pelkästään piirtoalue, joka on toteutettu samankaltaisesti pääsivun piirtoalueen kanssa. Lopullinen piirtoalueen väritys muuttuu hieman, kun yhteys on luotu Arduinoon (ks. luku 5).

4.3.3 Pääohjelman toiminta

Pääohjelman toteutus tehtiin kahdessa vaiheessa. Ensin ohjelmoitiin varsinainen datan vastaanotto Arduinolta ja sen lisääminen piirtoalueelle.

Toiseksi toteutettiin datan käsittelyyn tulleet ominaisuudet eli diskreetti Fourier-muunnos ja alipäästösuodatin.

Pääohjelmassa Connect-painiketta painettaessa lähetetään sarjaportin kautta viesti Arduinolle aloittaa AD-muunnos. Dataa vastaanotetaan reaaliajassa, ja jokainen saapunut AD-muuntimen tulos muutetaan volteiksi välille 0–5. Jokaiselle muunnokselle annetaan myös aika-arvo taustalla käynnissä olevasta ajastimesta. Tämän jälkeen jokainen [voltti, aika]-piste piirretään piirrosalueelle. Sarjaportin lukemisella ja pisteiden piirtämisellä on molemmilla omat prosessinsa.

Suorituskyvyn parantamiseksi piirrosaluetta päivitetään ajastimen avulla 100 ms:n välein eli 10 kertaa sekunnissa. Piirrosalue ei tämän takia näytä kaikkia pisteitä kerralla, mutta ne on mahdollista saada esiin hiiren oikealla napilla tulevasta valikosta valittaessa ”Piirrä kaikki pisteet”. Päivityksen aikana piirrosalue myös skaalataan uudelleen, jottei signaali mene alueen ulkopuolelle.

Datan vastaanottaminen voidaan lopettaa painamalla Disconnect-painiketta. Tämän jälkeen piirrettyä kuvaajaa voidaan lähentää, loitontaa, tallentaa kuvana tai kopioida. Kuvaajalle tai sen osalle voidaan myös tehdä alipäästösuodatus tai siitä voidaan laskea diskreetti Fourier-muunnos.

4.3.4 Digitaalisen suodattimen valinta

Työssä käytettiin eksponentiaalisesti liukuvan keskiarvon (exponentially moving average) alipäästösuodatinta, joka on IIR-tyypin suodatin. Alipäästösuodattimen avulla signaalista voidaan suodattaa korkean taajuuden komponentit pois. Eksponentiaalisesti liukuvan keskiarvon suodatin vastaa tyypiltään analogista vastuksen ja kondensaattorin suodatinta eli RC-suodatinta (resistor-capacitor). Eksponentiaalisesti liukuvan keskiarvon suodattimen tulos perustuu kaavaan 1. (Helpful 2013.)

$$s[i] = \propto \cdot x[i] + (1-\propto) \cdot s[i-1] \quad (1)$$

Kaavassa 1 *s[i]* kuvaa laskettavaa suodatettua tulosta ja *x[i]* signaalista otettua arvoa ajanhetkellä *i*. Sekä arvoa *x[i]* että edellistä suodatettua tulosta *s[i-1]* painotetaan *α*-arvolla, joka lasketaan kaavalla 2. (Helpful 2013.)

$$\propto = \frac{\Delta t}{RC+\Delta t} \tag{2}$$

Kaavassa 2 Δt on kahden näytteen välistä aikaa. *RC*:llä kuvataan aikaa, jossa analogisen RC-suodattimen kondensaattorin jännite latautuu takaisin 63 %:iin. *RC* saadaan laskettua kaavalla 3, jos tiedetään alipäästöraja f_c. (Helpful 2013.)

$$RC = \frac{1}{f_c \cdot 2 \cdot \pi} \tag{3}$$

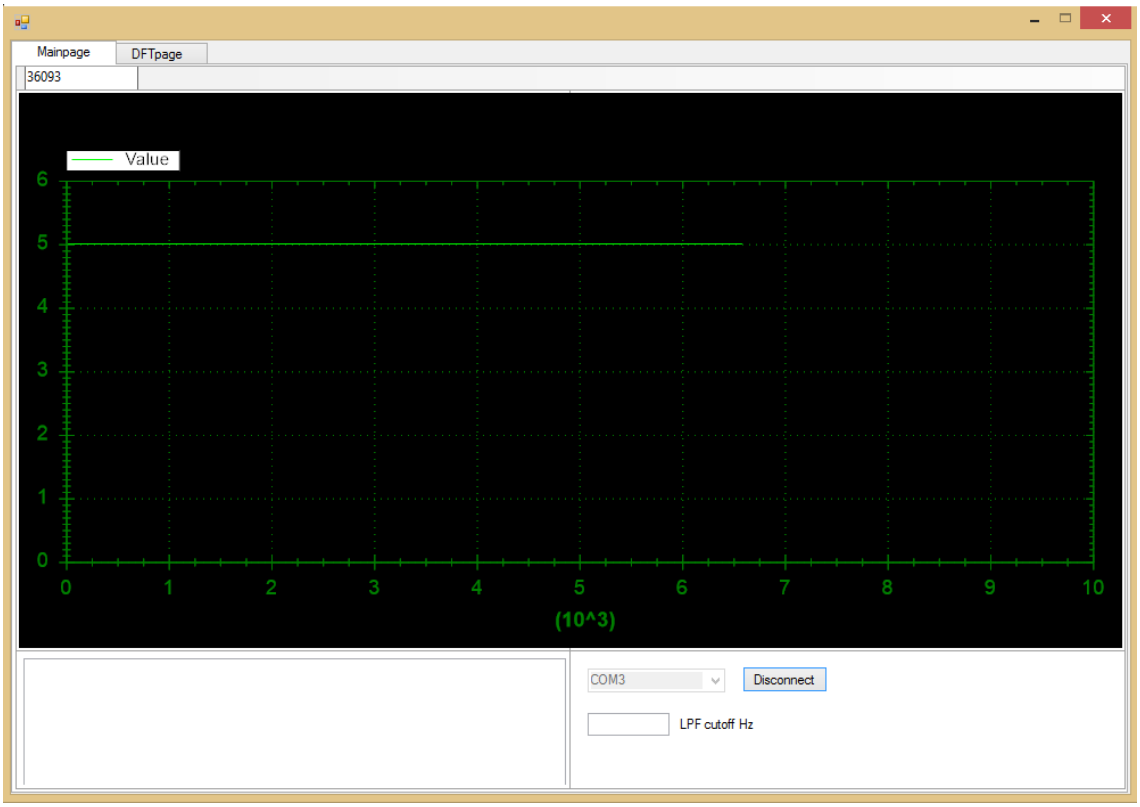
Eksponentiaalisesti liukuvan keskiarvon suodatin on hyvin yksinkertainen malli digitaalisesta suodattimesta (Helpful 2013.). Sillä voidaan kuitenkin helposti suodattaa esimerkiksi Nyquistin taajuuden jälkeisiä komponentteja pois.

5 TESTAUS

Työssä toteutetun signaalinkäsittelijän tärkeimpiä ominaisuuksia ovat sen näytteistysnopeus ja diskreetti Fourier-muunnos. Molempia ominaisuuksia testattiin yksinkertaisilla testitapauksilla. Ensimmäisessä testissä Arduino Uno R3:lta lähetettiin 5 V:n signaalia tietokoneelle, jonka näytteistysnopeudesta laskettiin keskiarvo. Toisessa testissä Uno R3:lta lähetettiin kanttiaaltoa tietokoneelle, jolla käsiteltiin signaalia.

5.1 Näytteistysnopeuden testaus

Arduino Uno R3:ssa on 5 V:n pinni, josta saatiin tasainen jännite testausta varten. Pinni yhdistettiin johtimella analogiseen pinniin A0, jonka jälkeen yhteys luotiin tietokoneen ja Arduinon välille. Käyttöliittymässä ylhäällä olevaa näytteistysnopeuden tekstikenttää päivitettiin 1 s:n välein, ja kentän arvo talletettiin muistiin. Signaalia vastaanotettiin 30 s:n ajan, ja tuloksista laskettiin keskiarvo. Kuvassa 10 näkyy kuvankaappaus testistä ja käyttöliittymän ulkonäöstä ajon aikana.



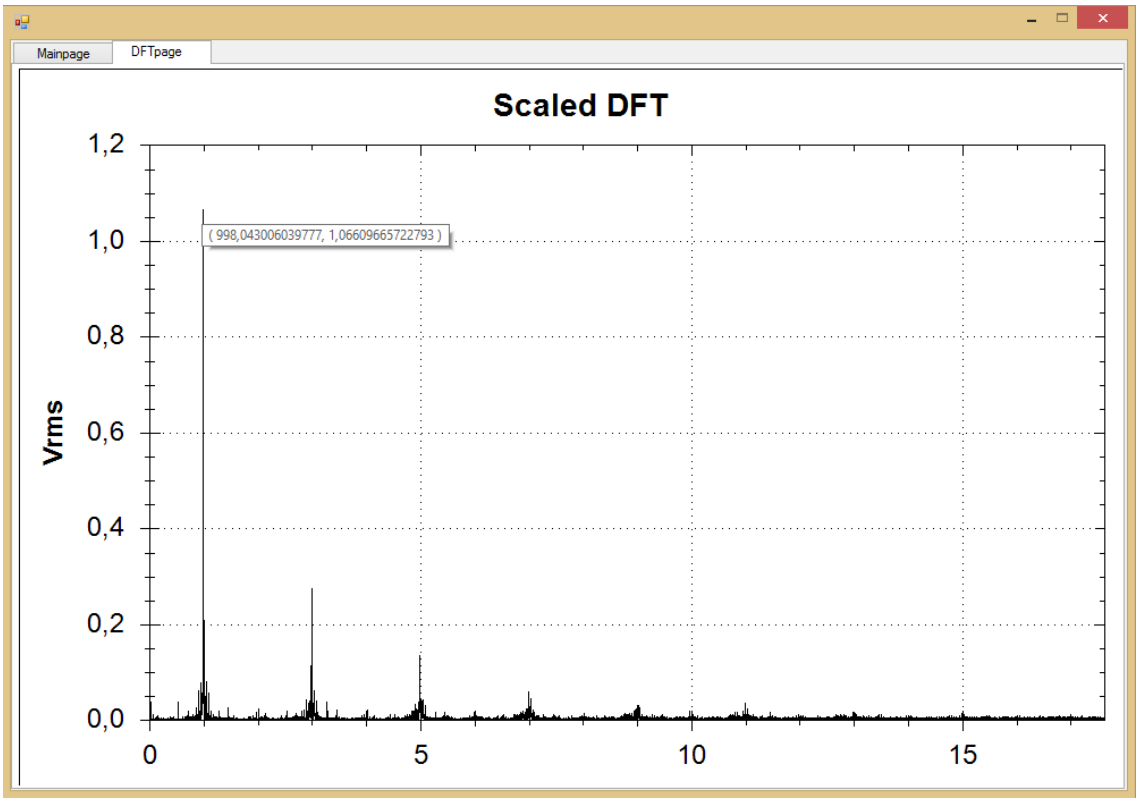
Kuva 10. 5 V:n signaalin kuvaaja.

Piirtoalueen alalaidassa näkyy kulunut aika millisekunteina ja vasemmassa laidassa voltit. Testin perusteella ohjelma pystyy vastaanottamaan ja piirtämään dataa noin 35 050 näytettä/s. Tämä on hyvin lähellä aikaisemmin luvussa 4.3.2 saatua arvoa 35 200 näytettä/s.

5.2 Diskreetin Fourier-muunnoksen testaus

Diskreettiä Fourier-muunnosta testattiin lähettämällä Arduino Uno R3:sta kanttiaaltoa tietyllä nopeudella. Samalla testattiin alipäästösuodattimen toimintaa. Arduinolla saadaan tehtyä kanttiaalto funktiolla tone(). Funktioon asetetaan muuttujiksi pinni ja taajuus, joista funktio tekee kanttiaallon 50 %:n pulssisuhteella (50 % ajasta 0 ja 50 % 1). Testissä käytettiin digitaalista pinniä 2, joka yhdistettiin pinniin A0 ja 1 kHz:n taajuutta. Kanttiaaltoa lähetettiin tietokoneelle 5 s:n ajan, josta otettiin 1 s:n osa muunnoksen laskemiseen.

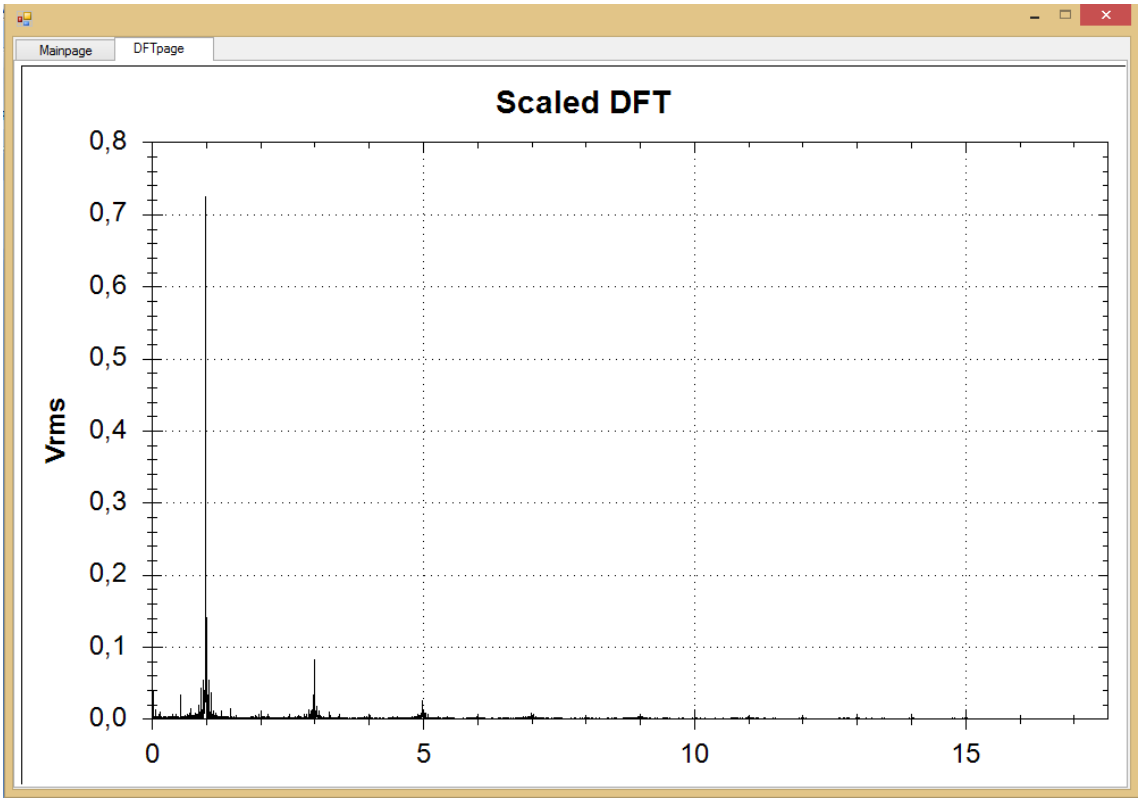
Kuvassa 11 ovat saadut tulokset kuvaajalla, jossa alareunassa näkyy taajuus kilohertseinä ja vasemmassa reunassa voltit.



Kuva 11. Diskreetin Fourier-muunnoksen kuvaaja 1 kHz:n kanttiaallosta.

Kuvassa 11 näkyy selkeä piikki noin 1 kHz:n kohdalla, mikä olikin odotettu tulos. Loput piikit johtuvat kanttiaallon harmonisesta värähtelystä eri taajuuksilla. Piirrosalue on myös mahdollista skaalata desibeliasteikolle, mutta se ei ollut tässä tapauksessa tarpeellista.

Samalla testattiin myös alipäästösuodattimen toimintaa. Harmonista värähtelyä suodatettiin asettamalla ”LPF cutoff Hz”-tekstikenttään arvo 1 100. Kuvassa 12 on suodatuksen jälkeen saadut tulokset.



Kuva 12. Diskreetti Fourier-muunnos suodatetusta 1 kHz:n kanttiaallosta.

Tulosten perusteella alipäästösuodatin ei ollut täysin ideaali mutta ensimmäisen asteen kompleksisuuteensa nähden kuitenkin hyvä. Se säilytti hyvin halutun 1 kHz:n taajuuden, mutta jätti hieman liikaa muita taajuuksia. Tulosta olisi helppo parantaa esimerkiksi lisäämällä alipäästösuodattimen rekursiivisuutta toiseen tai useampaan asteeseen. Tulokseen vaikuttaa myös kanttiaallon käyttö, joten esimerkiksi tavallista sinimuotoista signaalia suodattaessa päästään selvästi parempiin tuloksiin.

6 YHTEENVETO

Opinnäytetyön tavoitteena oli toteuttaa harrastelijakäyttöön sopiva digitaalinen signaalinkäsittelijä Arduinolla. Tähän tavoitteeseen päästiin ja tuloksena saatiin toimiva järjestelmä, joka saattaa omissa projekteissa tulla tulevaisuudessa hyvinkin hyödylliseksi. Diskreetin Fourier-muunnoksen onnistuminen oli positiivinen lopputulos. Sitä voidaan hyödyntää omissa tulevissa projekteissa esimerkiksi erilaisen kohinan tarkasteluun signaalissa. Kohinasta päästään eroon vielä paremmin lisäämällä tehokkaampia digitaalisia suodattimia ohjelmistoon.

Koska Arduinolla tehdään vain AD-muunnos ja lähetys tietokoneelle, on järjestelmää helppo kehittää eteenpäin. Tietokoneelle pystytään ohjelmallisesti lisäämään uusia toiminnallisuuksia digitaaliseen signaalinkäsittelyyn.

Seuraava kehityskohde tulee kuitenkin olemaan Arduinon ominaisuuksien kasvattaminen. Elektroniikkaprojekteissa tarvitaan usein suurempia virran- ja jännitteenkestoja kuin Uno pystyy tarjoamaan. Tämä vaatii ehdottomasti siirtymistä analogisten komponenttien käyttöön. Myös siirtymistä tehokkaampaan mikrokontrolleriin täytyy harkita.

Työn aihe ja sen parissa työskentely oli mielenkiintoista. Työ tarjosi sopivasti haasteita eikä mitään ylitsepääsemätöntä tullut vastaan. Työn lopputuloksena saatiin hyvä työkalu, jota voi kehittää haluttuun suuntaan.

LÄHTEET

ALGLIB 2014. About ALGLIB. Viitattu 17.5.2014 http://www.alglib.net.

Arduino 2014a. Arduino Uno. Viitattu 7.5.2014 http://arduino.cc/en/Main/ArduinoBoardUno.

Arduino 2014b. begin(). Viitattu 7.5.2014 http://arduino.cc/en/Serial/Begin.

Arduino 2014c. Introduction. Viitattu 7.5.2014 http://arduino.cc/en/Guide/Introduction.

Arduino 2014d. Software. Viitattu 7.5.2014 http://arduino.cc/en/Main/Software.

Atmel 2006. AVR120: Characterization and Calibration of the ADC on an AVR. Viitattu 12.5.2014 http://www.atmel.com/Images/doc2559.pdf.

Atmel 2009. 8-bit AVR Microcontroller with 4/8/16/32K Bytes In-System Programmable Flash. Viitattu 12.5.2014 http://www.atmel.com/Images/doc8161.pdf.

Barrett, S. 2013. Arduino Microcontroller: Processing for Everyone!. 3. painos. Morgan & Claypool Publishers.

Eclipse.baeyens.it 2014. Arduino eclipse plugin FAQ. Viitattu 17.5.2014 http://eclipse.baeyens.it/Arduino%20eclipse%20plugin%20FAQ.html.

Ifeachor, E.; Jervis, B. 2002. Digital Signal Processing: A Practical Approach. 2. painos. Harlow: Pearson Education Ltd.

Helpful 2013. Low-pass filter. Viitattu 18.5.2014 http://helpful.knobs-dials.com/index.php/Low-pass_filter.

Huttunen, H. 2014. Signaalinkäsittelyn perusteet. Tampere: Tampereen teknillinen yliopisto. Viitattu 2.5.2014 http://www.cs.tut.fi/kurssit/SGN-11000/SGN-11000.pdf.

Jokinen R. 2002. Digitaalinen signaalinkäsittely, lineaariset järjestelmät. Viitattu 2.5.2014 http://www.hovirinta.fi/audio/suunnittelu/teoriaa/DSP/Digitaalinen%20signaalink%E4sittely%2003.pdf.

Steven, W. 1997a. The Scientist and Engineer's Guide to Digital Signal Processing, Chapter 14: Introduction to Digital Filters. Viitattu 2.5.2014 http://www.dspguide.com/ch14/6.htm.

Steven, W. 1997b. The Scientist and Engineer's Guide to Digital Signal Processing, Chapter 9: Applications of the DFT. Viitattu 2.5.2014 http://www.dspguide.com/ch9/1.htm.

Visual Micro 2014. Arduino IDE for Visual Studio and Atmel Studio. Viitattu 18.5.2014 http://www.visualmicro.com.

Weisstein, E. 2014. Fast Fourier Transform. Mathworld. Viitattu 2.5.2014 http://mathworld.wolfram.com/FastFourierTransform.html.

ZedGraphWiki 2007. Main Page. Viitattu 18.5.2014 http://zedgraph.dariowiz.com.

Arduinon lopullinen ohjelmakoodi

```
// PS_16 = (1 << ADPS2);
// PS_32 = (1 << ADPS2) | (1 << ADPS0);
// PS_64 = (1 << ADPS2) | (1 << ADPS1);
// PS_128 = (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0);

volatile uint8_t incomingData;
volatile int flag;
volatile char incomingChar;

void setup()
{
  flag = 0;
  Serial.begin(1000000);
  tone(2, 500);

  cli(); //keskeytykset pois
  //alustetaan ADCSRA ja ADCSRB rekisterit
  ADCSRA = 0;
  ADCSRB = 0;
  DIDR0 |= (1 << ADC0D);
  ADMUX |= (1 << ADLAR); //ADC:n vasemmanpuoleinen tasaus - voidaan
lukea pelkkä ADCH
  ADCSRA |= (1 << ADEN); //ADC päälle
  ADCSRA |= (1 << ADPS2) | (1 << ADPS0); //ADC kellotaajuus -
16mHz/32=500kHz
  ADCSRA |= (1 << ADATE); //auto trigger
  sei();//keskeytykset päälle
}
//keskeytysten hallinta
ISR(ADC_vect) {
  flag = 1;//uusi luku saatu
}


void loop(){
  //luetaan käsky tietokoneelta ennen kuin aloitetaan keskeytykset
  if(Serial.available() > 0){
    incomingChar = Serial.read();
    //komento - aloita ADC:n käyttö
    if(incomingChar == 's') {
      ADCSRA |= (1 << ADIE); //ADC keskeytykset päälle
      ADCSRA |= (1 << ADSC); //aloita ADC arvojen muunto
    }
    //lopeta ADC:n käyttö
    else if(incomingChar == 'l') {
      ADCSRA &= ~(1 << ADIE);
      ADCSRA &= ~(1 << ADSC);
    }
  }
  if (flag == 1){
    incomingData = ADCH; //lue ADCH (0-255)
    Serial.write(incomingData); //lähetä tietokoneelle
    flag = 0; //luku lähetetty eteenpäin
  }
```

```
}
```

Arduinon testaukseen käytetty ohjelmakoodi

```
volatile uint8_t incomingData;
volatile int flag;
volatile long count;
volatile long starttime;

void setup()
{
  flag = 0;
  count = 0;
  Serial.begin(1000000);
  cli(); //keskeytykset pois
  ADCSRA = 0; //alustetaan ADCSRA ja ADCSRB rekisterit
  ADCSRB = 0;
  //DIDR0 |= (1 << ADC0D);
  ADMUX |= (1 << REFS0);
  ADMUX |= (1 << ADLAR); //voidaan lukea pelkkä ADCH
  ADCSRA |= (1 << ADEN); //ADC päälle
  ADCSRA |= (1 << ADPS2) | (1 << ADPS0); //16mHz/32=500kHz
  ADCSRA |= (1 << ADATE); //auto trigger
  ADCSRA |= (1 << ADIE); //ADC keskeytykset päälle
  ADCSRA |= (1 << ADSC); //aloita ADC arvojen muunto
  sei();//keskeytykset päälle
}
//keskeytysten hallinta
ISR(ADC_vect) {
  flag = 1;//uusi luku saatu
}
void loop(){
  if(count == 0) {starttime = millis();}
  if(count < 10000){
    if (flag == 1){
      incomingData = ADCH; //lue ADCH (0-255)
      Serial.write(incomingData); //lähetä tietokoneelle
      count++;
      flag = 0; //luku lähetetty eteenpäin
    }
  }
  else {
    long stoptime = millis();
    Serial.println(stoptime - starttime);
    count = 0;
    starttime = 0;
    delay(5000);
  }
}
```